

# Package: Sieve (via r-universe)

October 13, 2024

**Type** Package

**Title** Nonparametric Estimation by the Method of Sieves

**Version** 2.1

**Date** 2023-10-19

**Author** Tianyu Zhang

**Maintainer** Tianyu Zhang <tianyuz3@andrew.cmu.edu>

**Description** Performs multivariate nonparametric regression/classification by the method of sieves (using orthogonal basis). The method is suitable for moderate high-dimensional features (dimension < 100). The  $\ell_1$ -penalized sieve estimator, a nonparametric generalization of Lasso, is adaptive to the feature dimension with provable theoretical guarantees. We also include a nonparametric stochastic gradient descent estimator, Sieve-SGD, for online or large scale batch problems. Details of the methods can be found in: [arXiv:2206.02994](https://arxiv.org/abs/2206.02994) [arXiv:2104.00846](https://arxiv.org/abs/2104.00846) [arXiv:2310.12140](https://arxiv.org/abs/2310.12140).

**License** GPL-2

**Imports** Rcpp, combinat, glmnet, methods, MASS

**LinkingTo** Rcpp, RcppArmadillo

**RoxygenNote** 7.2.3

**Encoding** UTF-8

**NeedsCompilation** yes

**Repository** CRAN

**Date/Publication** 2023-10-19 14:10:02 UTC

## Contents

Sieve-package . . . . .	2
clean_up_result . . . . .	4
create_index_matrix . . . . .	4
GenSamples . . . . .	5

sieve.sgd.predict . . . . .	6
sieve.sgd.preprocess . . . . .	7
sieve.sgd.solver . . . . .	9
sieve_predict . . . . .	10
sieve_preprocess . . . . .	11
sieve_solver . . . . .	14

<b>Index</b>	<b>16</b>
--------------	-----------

---

Sieve-package	<i>Nonparametric Estimation by the Method of Sieves</i>
---------------	---

---

## Description

Performs multivariate nonparametric regression/classification by the method of sieves (using orthogonal basis). The method is suitable for moderate high-dimensional features (dimension  $< 100$ ). The  $l_1$ -penalized sieve estimator, a nonparametric generalization of Lasso, is adaptive to the feature dimension with provable theoretical guarantees. We also include a nonparametric stochastic gradient descent estimator, Sieve-SGD, for online or large scale batch problems. Details of the methods can be found in: [arXiv:2206.02994](https://arxiv.org/abs/2206.02994) [arXiv:2104.00846](https://arxiv.org/abs/2104.00846) [arXiv:2310.12140](https://arxiv.org/abs/2310.12140).

## Details

The DESCRIPTION file:

```

Package:      Sieve
Type:         Package
Title:        Nonparametric Estimation by the Method of Sieves
Version:      2.1
Date:         2023-10-19
Author:       Tianyu Zhang
Maintainer:   Tianyu Zhang <tianyuz3@andrew.cmu.edu>
Description:  Performs multivariate nonparametric regression/classification by the method of sieves (using orthogonal basis)
License:      GPL-2
Imports:      Rcpp, combinat, glmnet, methods, MASS
LinkingTo:    Rcpp, RcppArmadillo
RoxygenNote: 7.2.3
Encoding:     UTF-8

```

Index of help topics:

```

GenSamples      Generate some simulation/testing samples with
                 nonlinear truth.
Sieve-package   Nonparametric Estimation by the Method of
                 Sieves
clean_up_result Clean up the fitted model
create_index_matrix Create the index matrix for multivariate

```

	regression
sieve.sgd.predict	Sieve-SGD makes prediction with new predictors.
sieve.sgd.preprocess	Preprocess the original data for sieve-SGD estimation.
sieve.sgd.solver	Fit sieve-SGD estimators, using progressive validation for hyperparameter tuning.
sieve_predict	Predict the outcome of interest for new samples
sieve_preprocess	Preprocess the original data for sieve estimation.
sieve_solver	Calculate the coefficients for the basis functions

~~ An overview of how to use the ~~~ package, including the most ~~~ important functions ~~~

### Author(s)

Tianyu Zhang

Maintainer: Tianyu Zhang <tianyuz3@andrew.cmu.edu>

### References

Tianyu Zhang and Noah Simon (2022) <arXiv:2206.02994>

### Examples

```

xdim <- 5
basisN <- 1000
type <- 'cosine'

#non-linear additive truth. Half of the features are truly associated with the outcome
TrainData <- GenSamples(s.size = 300, xdim = xdim,
  frho = 'additive', frho.para = xdim/2)

#noise-free testing samples
TestData <- GenSamples(s.size = 1e3, xdim = xdim, noise.para = 0,
  frho = 'additive', frho.para = xdim/2)

sieve.model <- sieve_preprocess(X = TrainData[,2:(xdim+1)],
  basisN = basisN, type = type, interaction_order = 2)

sieve.model <- sieve_solver(sieve.model, TrainData$Y, l1 = TRUE)

sieve_model_prediction <- sieve_predict(testX = TestData[,2:(xdim+1)],
  testY = TestData$Y, sieve.model)

```

---

clean_up_result	<i>Clean up the fitted model</i>
-----------------	----------------------------------

---

**Description**

Clean up the fitted model

**Usage**

```
clean_up_result(sieve.model)
```

**Arguments**

sieve.model      a sieve sgd model.

**Value**

a processed sieve.model, adding function names and extract the best model

---

create_index_matrix	<i>Create the index matrix for multivariate regression</i>
---------------------	--

---

**Description**

Create the index matrix for multivariate regression

**Usage**

```
create_index_matrix(xdim, basisN = NULL, maxj = NULL, interaction_order = 5)
```

**Arguments**

xdim              a number. It specifies the predictors' dimension.  
basisN             a number. The number of basis function to use.  
maxj               a number. We use this to specify the largest row product in the index list.  
interaction\_order   a number The maximum order of interaction. 1 means additive model, 2 means including pairwise interaction terms, etc.

**Value**

a matrix. The first column is the product of the indices, the rest columns are the index vectors for constructing multivariate basis functions.

**Description**

This function is used in several examples in the package.

**Usage**

```
GenSamples(  
  s.size,  
  xdim = 1,  
  x.dis = "uniform",  
  x.para = NULL,  
  frho = "linear",  
  frho.para = 100,  
  y.type = "continuous",  
  noise.dis = "normal",  
  noise.para = 0.5  
)
```

**Arguments**

<code>s.size</code>	a number. Sample size.
<code>xdim</code>	a number. Dimension of the feature vectors $X$ .
<code>x.dis</code>	a string. It specifies the distribution of feature $X$ . The default is uniform distribution over $xdim$ -dimensional unit cube.
<code>x.para</code>	extra parameter to specify the feature distribution.
<code>frho</code>	a string. It specifies the true regression/log odds functions used to generate the data set. The default is a linear function.
<code>frho.para</code>	extra parameter to specify the true underlying regression/log odds function.
<code>y.type</code>	a string. Default is <code>y.type = 'continuous'</code> , meaning the outcome is numerical and the problem is regression. Set it to <code>y.type = 'binary'</code> for binary outcome.
<code>noise.dis</code>	a string. For the distribution of the noise variable (under regression problem settings). Default is Gaussian distribution.
<code>noise.para</code>	a number. It specifies the magnitude of the noise in regression settings.

**Value**

a data.frame. The variable  $Y$  is the outcome (either continuous or binary). Each of the rest of the variables corresponds to one dimension of the feature vector.

**Examples**

```

xdim <- 1 #1 dimensional feature
#generate 1000 training samples
TrainData <- GenSamples(s.size = 1000, xdim = xdim)
#generate some noise-free testing samples
TestData <- GenSamples(s.size = 1000, xdim = xdim, noise.para = 0)

```

---

```

sieve.sgd.predict      Sieve-SGD makes prediction with new predictors.

```

---

**Description**

Sieve-SGD makes prediction with new predictors.

**Usage**

```
sieve.sgd.predict(sieve.model, X)
```

**Arguments**

`sieve.model` a list initiated using `sieve.sgd.preprocess` and `sieve.sgd.solver`. Check the documentation of `sieve.sgd.preprocess` for more information.

`X` a data frame containing prediction features/ independent variables.

**Value**

`sieve.sgd.predict` will update the given `sieve.model` input list.

`inf.list` In each entry of the list `inf.list`, the array `prdy` is the predicted outcome under the given hyperparameter combination.

**Examples**

```

frho.para <- xdim <- 1 ##predictor dimension
frho <- 'additive' ###truth is a sum of absolute functions
type <- 'cosine' ###use cosine functions as the basis functions
#generate training data
TrainData <- GenSamples(s.size = 1e3, xdim = xdim,
                       frho.para = frho.para,
                       frho = frho, noise.para = 0.1)

#preprocess the model
sieve.model <- sieve.sgd.preprocess(X = TrainData[,2:(xdim+1)],
                                   type = type,
                                   s = c(1,2),
                                   r0 = c(0.5, 2, 4),
                                   J = c(1, 4, 8))

##train the model
sieve.model <- sieve.sgd.solver(sieve.model = sieve.model,

```

```

                                X = TrainData[,2:(xdim+1)],
                                Y = TrainData[,1])
##generate new data
NewData <- GenSamples(s.size = 5e2, xdim = xdim,
                     frho.para = frho.para,
                     frho = frho, noise.para = 0.1)
sieve.model <- sieve.sgd.predict(sieve.model, X = NewData[, 2:(xdim+1)])
plot(NewData[, 2:(xdim+1)], sieve.model$best_model$prdy)

```

---

sieve.sgd.preprocess *Preprocess the original data for sieve-SGD estimation.*

---

## Description

Preprocess the original data for sieve-SGD estimation.

## Usage

```

sieve.sgd.preprocess(
  X,
  s = c(2),
  r0 = c(2),
  J = c(1),
  type = c("cosine"),
  interaction_order = c(3),
  omega = c(0.51),
  norm_feature = TRUE,
  norm_para = NULL,
  lower_q = 0.005,
  upper_q = 0.995
)

```

## Arguments

- |    |  |
|----|--|
| X  | a data frame containing prediction features/ independent variables. The (i,j)-th element is the j-th dimension of the i-th sample's feature vector. So the number of rows equals to the sample size and the number of columns equals to the feature/covariate dimension. If the complete data set is large, this can be a representative subset of it (ideally have more than 1000 samples). |
| s  | numerical array. Smoothness parameter, a smaller s corresponds to a more flexible model. Default is 2. The elements of this array should take values greater than 0.5. The larger s is, the smoother we are assuming the truth to be.  |
| r0 | numerical array. Initial learning rate/step size, don't set it too large. The step size at each iteration will be $r0 * (\text{sample size})^{-1/(2s+1)}$ , which is slowly decaying.  |
| J  | numerical array. Initial number of basis functions, a larger J corresponds to a more flexible estimator. The number of basis functions at each iteration will be $J * (\text{sample size})^{1/(2s+1)}$ , which is slowly increasing. We recommend use  |

	J that is at least the dimension of predictor, i.e. the column number of the X matrix.
type	a string. It specifies what kind of basis functions are used. The default is (aperiodic) cosine basis functions ('cosine'), which is enough for generic usage.
interaction_order	a number. It also controls the model complexity. 1 means fitting an additive model, 2 means fitting a model allows, 3 means interaction terms between 3 dimensions of the feature, etc. The default is 3. For large sample size, lower dimension problems, try a larger value (but need to be smaller than the dimension of original features); for smaller sample size and higher dimensional problems, try set it to a smaller value (1 or 2).
omega	the rate of dimension-reduction parameter. Default is 0.51, usually do not need to change.
norm_feature	a logical variable. Default is TRUE. It means sieve_preprocess will rescale the each dimension of features to 0 and 1. Only set to FALSE when user already manually rescale them between 0 and 1.
norm_para	a matrix. It specifies how the features are normalized. For training data, use the default value NULL.
lower_q	lower quantile used in normalization. Default is 0.01 (1% quantile).
upper_q	upper quantile used in normalization. Default is 0.99 (99% quantile).

### Value

A list containing the necessary information for next step model fitting. Typically, the list is used as the main input of sieve.sgd.solver.

s.size.sofar	a number. Number of samples has been processed so far.
type	a string. The type of basis function.
hyper_para.list	a list of hyperparameters.
index.matrix	a matrix. Identifies the multivariate basis functions used in fitting.
index.row.prod	the index product for each basis function. It is used in calculating basis function - specific learning rates.
inf.list	a list storing the fitted results. It has a length of "number of unique combinations of the hyperparameters". The component of inf.list is itself a list, it has a hyper_para.index domain to specify its corresponding hyperparameters (need to be used together with hyper_para.list). Its rolling.cv domain is the progressive validation statistics for hyperparameter tuning; beta.f is the regression coefficients for the first length(beta.f) basis functions, the rest of the basis have 0 coefficients.
norm_para	a matrix. It records how each dimension of the feature/predictor is rescaled, which is useful when rescaling the testing sample's predictors.

**Examples**

```

xdim <- 1 #1 dimensional feature
#generate 1000 training samples
TrainData <- GenSamples(s.size = 1000, xdim = xdim)
sieve.model <- sieve.sgd.preprocess(X = TrainData[,2:(xdim+1)])

```

---

sieve.sgd.solver	<i>Fit sieve-SGD estimators, using progressive validation for hyperparameter tuning.</i>
------------------	--

---

**Description**

Fit sieve-SGD estimators, using progressive validation for hyperparameter tuning.

**Usage**

```
sieve.sgd.solver(sieve.model, X, Y, cv_weight_rate = 1)
```

**Arguments**

sieve.model	a list initiated using sieve.sgd.preprocess. Check the documentation of sieve.sgd.preprocess for more information.
X	a data frame containing prediction features/ independent variables.
Y	training outcome.
cv_weight_rate	this governs the divergence rate of rolling validation statistics. Default is set to be 1 and in general does not need to be changed.

**Value**

A list. It contains the fitted regression coefficients and progressive validation statistics for each hyperparameter combination.

s.size.sofar	a number. Number of samples has been processed so far.
type	a string. The type of basis function.
hyper.para.list	a list of hyperparameters.
index.matrix	a matrix. Identifies the multivariate basis functions used in fitting.
index.row.prod	the index product for each basis function. It is used in calculating basis function - specific learning rates.
inf.list	a list storing the fitted results. It has a length of "number of unique combinations of the hyperparameters". The component of inf.list is itself a list, it has a hyper.para.index domain to specify its corresponding hyperparameters (need to be used together with hyper.para.list). Its rolling.cv domain is the progressive validation statistics for hyperparameter tuning; beta.f is the regression coefficients for the first length(beta.f) basis functions, the rest of the basis have 0 coefficients.
norm_para	a matrix. It records how each dimension of the feature/predictor is rescaled, which is useful when rescaling the testing sample's predictors.

**Examples**

```

frho.para <- xdim <- 1 ##predictor dimension
frho <- 'additive' ###truth is a sum of absolute functions
type <- 'cosine' ###use cosine functions as the basis functions
#generate training data
TrainData <- GenSamples(s.size = 1e3, xdim = xdim,
                        frho.para = frho.para,
                        frho = frho, noise.para = 0.1)

#preprocess the model
sieve.model <- sieve.sgd.preprocess(X = TrainData[,2:(xdim+1)],
                                   type = type,
                                   s = c(1,2),
                                   r0 = c(0.5, 2, 4),
                                   J = c(1, 4, 8))

##train the model
sieve.model <- sieve.sgd.solver(sieve.model = sieve.model,
                               X = TrainData[,2:(xdim+1)],
                               Y = TrainData[,1])

##sieve-SGD can do multiple passes over the data, just like other SGD methods.
##usually a second pass can still improve the prediction accuracy
##watch out overfitting when performing multiple passes!
sieve.model <- sieve.sgd.solver(sieve.model = sieve.model,
                               X = TrainData[,2:(xdim+1)],
                               Y = TrainData[,1])

```

---

sieve\_predict

*Predict the outcome of interest for new samples*


---

**Description**

Use the fitted sieve regression model from `sieve_solver`. It also returns the testing mean-squared errors.

**Usage**

```
sieve_predict(model, testX, testY = NULL)
```

**Arguments**

<code>model</code>	a list. Use the fitted model from <code>sieve_solver</code> .
<code>testX</code>	a data frame. Dimension equals to test sample size x feature diemnsion. Should be of a similar format as the training feature provided to <code>sieve_preprocess</code> .
<code>testY</code>	a vector. The outcome of testing samples (if known). Default is <code>NULL</code> . For regression problems, the algorithm also returns the testing mean-squared errors.

**Value**

a list.

`predictY` a matrix. Dimension is test sample size (# of rows) x number of penalty hyperparameter lambda (# of columns). For regression problem, that is, when `family = "gaussian"`, each entry is the estimated conditional mean (or predictor of outcome `Y`). For classification problems (`family = "binomial"`), each entry is the predicted probability of having `Y = 1` (which class is defined as "class 1" depends on the training data labeling).

`MSE` For regression problem, when `testY` is provided, the algorithm also calculates the mean-squared errors using testing data. Each entry of `MSE` corresponds to one value of penalization hyperparameter `lambda`

**Examples**

```
xdim <- 1 #1 dimensional feature
#generate 1000 training samples
TrainData <- GenSamples(s.size = 1000, xdim = xdim)
#use 50 cosine basis functions
type <- 'cosine'
basisN <- 50
sieve.model <- sieve_preprocess(X = TrainData[,2:(xdim+1)],
                              basisN = basisN, type = type)
sieve.fit <- sieve_solver(model = sieve.model, Y = TrainData$Y)
#generate 1000 testing samples
TestData <- GenSamples(s.size = 1000, xdim = xdim)
sieve.prediction <- sieve_predict(model = sieve.fit,
                                 testX = TestData[,2:(xdim+1)],
                                 testY = TestData$Y)

###if the outcome is binary,
###need to solve a nonparametric logistic regression problem
xdim <- 1
TrainData <- GenSamples(s.size = 1e3, xdim = xdim, y.type = 'binary', frho = 'nonlinear_binary')
sieve.model <- sieve_preprocess(X = TrainData[,2:(xdim+1)],
                              basisN = basisN, type = type)
sieve.fit <- sieve_solver(model = sieve.model, Y = TrainData$Y,
                        family = 'binomial')

###the predicted value is conditional probability (of taking class 1).
TrainData <- GenSamples(s.size = 1e3, xdim = xdim, y.type = 'binary', frho = 'nonlinear_binary')
sieve.prediction <- sieve_predict(model = sieve.fit,
                                 testX = TestData[,2:(xdim+1)])
```

---

sieve\_preprocess

*Preprocess the original data for sieve estimation.*


---

**Description**

Generate the design matrix for the downstream lasso-type penalized model fitting.

**Usage**

```
sieve_preprocess(
  X,
  basisN = NULL,
  maxj = NULL,
  type = "cosine",
  interaction_order = 3,
  index_matrix = NULL,
  norm_feature = TRUE,
  norm_para = NULL
)
```

**Arguments**

<code>X</code>	a data frame containing original features. The (i,j)-th element is the j-th dimension of the i-th sample's feature vector. So the number of rows equals to the sample size and the number of columns equals to the feature dimension.
<code>basisN</code>	number of sieve basis function. It is in general larger than the dimension of the original feature. Default is 50*dimension of original feature. A larger value has a smaller approximation error but it is harder to estimate. The computational time/memory requirement should scale linearly to <code>basisN</code> .
<code>maxj</code>	a number. the maximum index product of the basis function. A larger value means more <code>basisN</code> . If <code>basisN</code> is already specified, do not need to provide value for this argument.
<code>type</code>	a string. It specifies what kind of basis functions are used. The default is (aperiodic) cosine basis functions, which is suitable for most purpose.
<code>interaction_order</code>	a number. It also controls the model complexity. 1 means fitting an additive model, 2 means fitting a model allows, 3 means interaction terms between 3 dimensions of the feature, etc. The default is 3. For large sample size, lower dimension problems, try a larger value (but need to be smaller than the dimension of original features); for smaller sample size and higher dimensional problems, try set it to a smaller value (1 or 2).
<code>index_matrix</code>	a matrix. provide a pre-generated index matrix. The default is <code>NULL</code> , meaning <code>sieve_preprocess</code> will generate one for the user.
<code>norm_feature</code>	a logical variable. Default is <code>TRUE</code> . It means <code>sieve_preprocess</code> will rescale the each dimension of features to 0 and 1. Only set to <code>FALSE</code> when user already manually rescale them between 0 and 1.
<code>norm_para</code>	a matrix. It specifies how the features are normalized. For training data, use the default value <code>NULL</code> .

**Value**

A list containing the necessary information for next step model fitting. Typically, the list is used as the main input of `Sieve::sieve_solver`.

Phi	a matrix. This is the design matrix directly used by the next step model fitting. The (i,j)-th element of this matrix is the evaluation of i-th sample's feature at the j-th basis function. The dimension of this matrix is sample size x basisN.
X	a matrix. This is the rescaled original feature/predictor matrix.
type	a string. The type of basis function.
index_matrix	a matrix. It specifies what are the product basis functions used when constructing the design matrix Phi. It has a dimension basisN x dimension of original features. There are at most interaction_order many non-1 elements in each row.
basisN	a number. Number of sieve basis functions.
norm_para	a matrix. It records how each dimension of the feature/predictor is rescaled, which is useful when rescaling the testing sample's predictors.

## Examples

```

xdim <- 1 #1 dimensional feature
#generate 1000 training samples
TrainData <- GenSamples(s.size = 1000, xdim = xdim)
#use 50 cosine basis functions
type <- 'cosine'
basisN <- 50
sieve.model <- sieve_preprocess(X = TrainData[,2:(xdim+1)],
                               basisN = basisN, type = type)
#sieve.model$Phi #Phi is the design matrix

xdim <- 5 #1 dimensional feature
#generate 1000 training samples
#only the first two dimensions are truly associated with the outcome
TrainData <- GenSamples(s.size = 1000, xdim = xdim,
                       frho = 'additive', frho.param = 2)

#use 1000 basis functions
#each of them is a product of univariate cosine functions.
type <- 'cosine'
basisN <- 1000
sieve.model <- sieve_preprocess(X = TrainData[,2:(xdim+1)],
                               basisN = basisN, type = type)
#sieve.model$Phi #Phi is the design matrix

#fit a nonparametric additive model by setting interaction_order = 1
sieve.model <- sieve_preprocess(X = TrainData[,2:(xdim+1)],
                               basisN = basisN, type = type,
                               interaction_order = 1)
#sieve.model$index_matrix #for each row, there is at most one entry >= 2.
#this means there are no basis functions varying in more than 2-dimensions
#that is, we are fitting additive models without interaction between features.

```

sieve\_solver

*Calculate the coefficients for the basis functions***Description**

This is the main function that performs sieve estimation. It calculate the coefficients by solving a penalized lasso type problem.

**Usage**

```
sieve_solver(
  model,
  Y,
  l1 = TRUE,
  family = "gaussian",
  lambda = NULL,
  nlambda = 100
)
```

**Arguments**

model	a list. Typically, it is the output of Sieve::sieve_preprocess.
Y	a vector. The outcome variable. The length of Y equals to the training sample size, which should also match the row number of X in model.
l1	a logical variable. TRUE means calculating the coefficients by solving a l1-penalized empirical risk minimization problem. FALSE means solving a least-square problem. Default is TRUE.
family	a string. 'gaussian', mean-squared-error regression problem.
lambda	same as the lambda of glmnet::glmnet.
nlambda	a number. Number of penalization hyperparameter used when solving the lasso-type problem. Default is 100.

**Value**

a list. In addition to the preprocessing information, it also has the fitted value.

Phi	a matrix. This is the design matrix directly used by the next step model fitting. The (i,j)-th element of this matrix is the evaluation of i-th sample's feature at the j-th basis function. The dimension of this matrix is sample size x basisN.
X	a matrix. This is the rescaled original feature/predictor matrix.
beta_hat	a matrix. Dimension is basisN x nlambda. The j-th column corresponds to the fitted regression coefficients using the j-th hyperparameter in lambda.
type	a string. The type of basis funtion.

<code>index_matrix</code>	a matrix. It specifies what are the product basis functions used when constructing the design matrix Phi. It has a dimension <code>basisN</code> x dimension of original features. There are at most <code>interaction_order</code> many non-1 elements in each row.
<code>basisN</code>	a number. Number of sieve basis functions.
<code>norm_para</code>	a matrix. It records how each dimension of the feature/predictor is rescaled, which is useful when rescaling the testing sample's predictors.
<code>lambda</code>	a vector. It records the penalization hyperparameter used when solving the lasso problems. Default has a length of 100, meaning the algorithm tried 100 different penalization hyperparameters.
<code>family</code>	a string. 'gaussian', continuous numerical outcome, regression problem; 'binomial', binary outcome, classification problem.

### Examples

```

xdim <- 1 #1 dimensional feature
#generate 1000 training samples
TrainData <- GenSamples(s.size = 1000, xdim = xdim)
#use 50 cosine basis functions
type <- 'cosine'
basisN <- 50
sieve.model <- sieve_preprocess(X = TrainData[,2:(xdim+1)],
                              basisN = basisN, type = type)
sieve.fit<- sieve_solver(model = sieve.model, Y = TrainData$Y)

###if the outcome is binary,
###need to solve a nonparametric logistic regression problem
xdim <- 1
TrainData <- GenSamples(s.size = 1e3, xdim = xdim, y.type = 'binary', frho = 'nonlinear_binary')
sieve.model <- sieve_preprocess(X = TrainData[,2:(xdim+1)],
                              basisN = basisN, type = type)
sieve.fit<- sieve_solver(model = sieve.model, Y = TrainData$Y,
                        family = 'binomial')

```

# Index

\* **sieve estimation; estimation using  
orthogonal series**

Sieve-package, [2](#)

clean\_up\_result, [4](#)

create\_index\_matrix, [4](#)

GenSamples, [5](#)

Sieve (Sieve-package), [2](#)

Sieve-package, [2](#)

sieve.sgd.predict, [6](#)

sieve.sgd.preprocess, [7](#)

sieve.sgd.solver, [9](#)

sieve\_predict, [10](#)

sieve\_preprocess, [11](#)

sieve\_solver, [14](#)